MWSUG 2019 – Paper AL-078

Machine Learning? The Machine Already Knows. Software-Intelligent Application Development Provides Reliability, Reusability, Extendability, and Maintainability for Strong Smart Systems in an Ever-Changing World

LeRoy Bessler PhD
Bessler Consulting and Research, Mequon, WI, USA
Le_Roy_Bessler@wi.rr.com

Abstract

Applications designed and built with Software Intelligence (SI) are robust, made of reusable parts, and easy and quick to extend or maintain. With dynamically auto-customizing code, such "living" applications go beyond change tolerance to change amenability, and further—to change implementation. They cope with ever-changing user or management preferences, run-dates, data dates, and data content. Common types of changes in report/graph content, format, and function are handled without reprogramming. If business rules do not change, an SI application can have eternal life.

Whether you are a new or experienced SAS® programmer, or an analytically oriented user who does not think of herself or himself as a programmer at all, this paper—which assumes no advanced SAS knowledge—shows you how to apply principles of Software-Intelligent Application Development, which are really programming-language-independent, to make your use of SAS software safe, simple, and speedy.

One of the tools for SI implementation with SAS software is SAS macro language. For users with no SAS macro language experience, the standup presentation includes a brief, but sufficient, introduction.

Besides explaining the few, but powerful, principles of Software-Intelligent application development, the paper provides you with some widely applicable practical examples that you can put to use back at work.

Introduction

Software Intelligence permits application programs to dynamically customize themselves, without human intervention, to continue to meet design requirements in a changing environment. Rather than static expressions and engines of single-point-in-time programmer decision or user choice, such adaptive programs are “Living Applications”. They go beyond change amenability (maintainability) to change auto-implementation. Short of a revolution in host computer technical architecture, or in business or research process to be served, such applications have the gift of “Eternal Life”.

This paper explains how the objectives of reliability, reusability, extendability, and maintainability can be met with the SAS System by using Software Intelligence (SI), and

illustrates that with examples. Parameter files, macro variables, and macros are SI enablers for applications that do dynamic auto-customization (i.e., that modify themselves).

Some prior implementations of dynamically auto-customized applications—that can handle, from run to run, the vicissitudes of data and/or date while meeting report or graph format and function requirements, and that adapt to changing user or management preferences—were documented in a series of papers. (Please see the References.) This paper assumes basic understanding of SAS macro language.

Five coding examples are provided in the Appendix. Example 5 is derived from an interesting application of Software-Intelligent Application Development presented in Reference 5.

Software-Intelligent Application Development

SI application development is necessary for Maturation in SAS software Use. The first stage of SAS software use is as an end-user tool—for ad hoc data analysis or data presentation tasks. The second stage is when the site's SAS support staff enhance SAS software as an enterprise-wide utility, by providing site-specific customization, macros, formats, templates, etc. The ultimate stage is SAS as a production application development tool, whether for on-demand online/web systems or scheduled (possibly computer-scheduled) batch processing systems.

Ad hoc processing is typically: (a) one-time or irregularly needed; (b) custom and iterative in development of the program code; and (c) often done interactively. Production processing, online/web or batch, is standardized and hands-off. It must get everything right the first time, every time.

Reliability

One reliability policy is simple—once your program is working right, never touch it again. *The only safe program change is no change.*

More hazardous than changing your own long-in-service program is to change one that someone else wrote. Most hazardous is to change a program that several people have maintained. Part of an old program may even be doing no longer needed processing and producing no longer referenced outputs. Such refusal or neglect to maintain the program is a tacit admission that no change is a safe change.

But, since user needs do change, an application program must change to meet them. What is a reasonable recourse?

Foreseeable change can be, and is best, supported through Software Intelligence.

For example, if a tabular or graphic report takes as input the last N years, months, weeks, or days of history, it is more prudent to keep N in a parameter file that is read by the program, rather than “hard code” N in the program itself. Every time you or a successor might open the program to change N, the program would be at risk.

Another good candidate to store in a parameter file is a goal or threshold for a measurement variable. Since judgment of what is good or bad changes over time, it is best to plan to accommodate that without program change.

Common for some application developers, especially if not full-time IT professionals, is to include data in the program. This is a productivity aid during application development and debugging, when you do not want to go to a separate file to change the input every time you need to test a different case. However, when development is complete, data should be separated from the program.

Too frequent in applications, especially if originally written for a supposed one-time analysis or report (any ad-hoc application, if valuable, is likely to experience recurrent use), are manually entered dates, for a title and/or for a filter on data selection. If such a date is dependably a function of run-date, let the program use SAS functions to retrieve today's date and to compute and construct the title or filter date(s) from it. If not a function of run-date, supply the manually entered date via a parameter file, for the program to read.

Program-change avoidance (i.e., reliability enhancement) is implemented in the situations described above by what I call “Building Firewalls”. **Build Firewalls** between your program and the data, between your program and your (and everyone else’s) programming keyboard.

(My first use of “firewall” in the context of this discussion long antedated the internet, which requires a different kind of firewall.)

With parameter files, macro variables, and macros, SI can protect program integrity, but still support limited revision of format, content, or function—to support a “flexible freeze” (to borrow a phrase from the 1970’s USA-Russia nuclear weapons control dialogue).

Reusability

Reusability can be implemented with includable blocks of source code, or macros. Code or macros are best stored in shared-access libraries so that anyone, furnished with documentation as to availability, purpose, required inputs, and provided outputs, can use it or them. Unfortunately, including reusable code by saying “%INCLUDE *sourcefilename*” fails to disclose what the reusable code’s inputs and outputs are. Invocation of a well-designed macro, however, can require the explicit identification of the names of the inputs and outputs via assignment of values to parameters. Other parameters are used to control the function of the macro. Such a macro “documents” the program, and is less likely to be erroneously invoked when reused.

Extendability

When I wrote the first edition of a Visual Information System prototype, every time the number of graph selections on the menu changed (typically, increased), I had to change lots of program code. Eventually, I restructured the application with macro processing, and controlled the number of graph selections via a macro parameter, supplied “outside” from a SAS AUTOEXEC

file. This provided extendability (or shrinkability) by requiring the change of only one number, and protected the working program code.

The benefit was non-trivial. Prior to the extendable macro implementation, each selection line required its own screen definition code, its own response-field initialization and editing code, etc. The macro's Software-Intelligent design dynamically auto-customized the application, without reprogramming every time the user needs changed.

More recently, my colleague Dr. Francesca Pierri and I updated for the web and email an Enterprise Performance Reporting system that I had developed back in 1987. The structure (exception reporting for the current report period, exception history reporting, summary reporting for the current report period, and trend reporting) for each performance criterion is made externally controllable via a table of descriptions, variable names, standards, types of standards (comparison operators), and formats for the standard and actual values. When another performance measure is established, one simply adds another row to the control table. Extending the system requires no program change. Likewise, changing the standard for a performance measure requires no program change—which ties into Maintainability, to be discussed next. The use of such a control table is demonstrated in Example 5.

The Reality of Maintainability

As a programmer for thirty-five years, I have read and heard various claims about tools and methods (I am pleased, at least, that the preposterous puffery of needless polysyllabification "methodology" and "methodologies" has gone out of style) that were guaranteed to make it easy to maintain code. Here are my early conclusions about this challenge.

Bessler's First Theorem: Application maintenance is easy only when maintainer and creator are the same person.

Bessler's Corollary to the First Theorem: Ease of Maintenance, E, is inversely proportional to c raised to the power N-1, where N is the number of persons who have written or modified the code. Count the creator in N. (One *might* conjecture that N should instead count the creation event and the maintenance events, not just the creator and distinct maintainers.)

The Corollary is mathematically expressed with this formula:

$E = mc^{N-1}$

c is a constant for which the exact value still must be discovered, but we do know with certainty that $1 > c > 0$ so that as the number of program maintainers (or number of program maintenance events) increases, the maintainability goes down.

The constant m is NOT mass and the constant c is NOT the speed of light,
even if my physicist heritage forced me to emulate this famous formula, $E = mc^2$.

Frankly, the value for m is arbitrary. It can be whatever you would like it to be the (arbitrarily chosen) measure of maximum of Ease of Maintenance. You might like 100, the perfect score when grading an examination. Or you might prefer 10, the customary maximum when ranking the goodness of some quality on a scale from 1 to 10.

Whenever it is the case that N is 1 (i.e., the maintainer is the person who was the creator, and there is this only this initial maintenance event), E is exactly equal to m, the Maximum Ease of Maintenance, because c (regardless of its value) raised to the power 0 is equal to 1.

Bessler's Second Theorem: Application maintenance is very easy only if the maintainer created it recently—within the last few weeks, preferably yesterday. When looking at an old program, even a very experienced programmer often must ask herself or himself: "Why did I do that?" I leave it to a future investigator to suggest a formula to quantify the effect described in my Second Theorem.

Examples

The Appendix contains five coding examples. Example 1 is the easiest way to use SAS macro language to apply simple, quick, safe changes to a program. You do not need to be a macro programmer for this. Example 2 is a "helping" macro to do simple auto-customized reporting. The remaining examples are bona fide examples of software intelligent applications. Examples 3 and 4 are macros to do dynamically auto-customized reporting. Example 5 combines a software-intelligent macro with a control table to demonstrate how to build an application where the program code will never need to be touched again. A recent interesting application of Software-Intelligent Application Development is presented in Reference 5.

Conclusion

Software Intelligence can make application maintenance rare, quick, and safe. All foreseeable changes and extensions can be best delivered by merely updating one or more parameter files, rather than by changing program code, or changing **`%LET`** statements at the top of a program.

References (Related Papers by the Author)

1. Intelligent Production Graphic Reporting Applications, *Proceedings of the Sixteenth Annual SAS Users Group International Conference*, 1991. Cary, NC: SAS Institute Inc.

2. Software Intelligence: Applications That Customize Themselves, *Proceedings of the Eighteenth Annual SAS Users Group International Conference*, 1993. Cary, NC: SAS Institute Inc.

3. Reusable, Extendable, Maintainable, Reliable Application Development: Using Software Intelligence to Build an EIS with Only SAS & SAS/GRAPH® Software, *Proceedings of the Twentieth Annual SAS Users Group International Conference*, 1995. Cary, NC: SAS Institute Inc.

4. Strong, Smart Systems: Software-Intelligent Development for Reliable, Reusable, Extendable, and Maintainable Applications, *Proceedings of the Twenty-Third Annual SAS Users Group International Conference*, 1998. Cary, NC: SAS Institute Inc.

5. Tell Them What's Important: Communication-Effective Web- and Email-Based Software-Intelligent Enterprise Performance Reporting, *Proceedings of the Twenty-Eighth Annual SAS*

Users Group International Conference, 2003. Cary, NC: SAS Institute Inc. With Francesca Pierri PhD.

Author Information

Your questions, comments, and suggestions are always welcome.

LeRoy Bessler PhD
Mequon, Wisconsin, USA
Strong Smart Systems™
Visual Data Insights™
Le_Roy_Bessler@wi.rr.com

Dr. LeRoy Bessler has presented at conferences in the USA, Canada, and Europe, on effective visual communication (using graphs, tables, web pages, maps, or color), SAS to Excel, tools for SAS server administrators, users, and managers, and Application Development for Reliability, Reusability, Maintainability, Extendibility, and Flexibility. His experience includes application development and supporting users, servers, software, and data. He is writing a book on communication-effective data visualization, and has declared himself to be a Data Artist.

Appendix: Examples

Example 1: Simplest Use of SAS Macro Language. Easy Way To Do Safe Program Changes

This illustration is ridiculous, in terms of the brevity and simplicity of the sample program. A more realistic program might be tens or hundreds of lines of code, or even thousands. Suppose that from run to run you need to change just one aspect of the program functions. E.g., do it for a different selection of the data.

If you are going to change a program at all, it is certainly convenient to change it in only one place, and, ideally, right at the top of the program, rather than having to hunt for the right location(s) in the code.

The SAS Macro Facility allows your program to store information in a symbol table in computer memory at run time, and allows your program to retrieve information from the symbol table. Here is a sample program and the resulting output.

```
%LET TitleText = Students 11 Years Old;
%LET AgeSelect = 11;
title "&TitleText";
proc print data=sashelp.class(where=(age EQ &AgeSelect)); run;
```

Students 11 Years Old

Obs	Name	Sex	Age	Height	Weight
6	Joyce	F	11	51.3	50.5
18	Thomas	M	11	57.5	85.0

The **%LET** statement is probably the easiest tool (except for **%PUT**) in the SAS macro language to master. It stores information in the symbol table. (You can also do so with CALL SYMPUT, but that is outside the scope of this discussion.)

As you can guess from above, the & is a signal to SAS that you want it to retrieve information with that symbolic name from the symbol table. (You can also do so with CALL SYMGET, but that is outside the scope of this discussion.)

Tip: If you are supplying a value that must appear within quotes in the final resolved run-time program, be sure to use double quotes, as in—

```
TITLE "&TitleText";
```

If single quotes were used in the TITLE statement, the above report would display

```
&TitleText
```

instead of

```
Students 11 Years Old
```

Example 2: A Simple SAS Macro. In the prior example, a macro variable was used in a data selection filter (i.e., to control the reported range of values). This example is the converse. It extracts from a date-keyed data set the earliest and latest dates (i.e., the boundaries of the range of found values), and supplies them as formatted global variables for use in a TITLE statement.

```
%MACRO DateRange(data=,
                 DateVar=,
                 FirstDateMacroVar=,
                 LastDateMacroVar=);

proc sort data=&DATA out=DATES(keep=&DateVar) nodupkeys;
by &DATEVAR;
run;

data _null_;
%global &FirstDateMacroVar &LastDateMacroVar;
set dates end=Last;
by &DATEVAR;
if _N_ EQ 1 then
call symput("&FirstDateMacroVar",
            trim(left(put(&DATEVAR,WORDDATE20.))));
else
if Last then
call symput("&LastDateMacroVar",
            trim(left(put(&DATEVAR,WORDDATE20.))));
run;

%MEND  DateRange;

%DateRange(data=SASHELP.CITIDAY,
           DateVar=DATE,
           FirstDateMacroVar=FirstDate,
           LastDateMacroVar=LastDate)
run;

title "Dow Jones from &FirstDate to &LastDate";
proc print data=SASHELP.CITIDAY(keep=SNYDJCM DATE);
format DATE YYMMDD10.;
run;
```

Output from PROC PRINT:

Dow Jones from January 1, 1988 to February 5, 1992

Obs	DATE	SNYDJCM
1	1988-01-01	.
2	1988-01-04	740.20
3	1988-01-05	747.38
...	...	...
1069	1992-02-05	1167.85

Example 3. Rank the Top NN observations from a data set, with NN being selectable and appearing automatically in the title. If the ranking includes all the observations, or if a minimum cutoff is used, then the title says "Ranked List of", rather than "Top NN". A subtitle shows what percent of the total is accounted for in the table. An extra subtitle is generated if the minimum cutoff is used with effect.

Output 3A:

```
Top 10 Students By Weight

This list accounts for 61.3% of the total Weight in Pounds

                                Weight
                                  in
                Rank    Student   Pounds

                  1     Philip    150.0
                  2     Ronald    133.0
                  3     Robert    128.0
                  4     Janet     112.5
                  5     Alfred    112.5
                  6     Mary      112.0
                  7     William   112.0
                  8     Carol     102.5
                  9     Henry     102.5
                 10     John       99.5
                                 ======
                                 1164.5
```

Program 3A:

```
%TOPNN(DATA=sashelp.class,
       CLASSVAR=Name, CVARLABL=Student,
       RANKVAR=Weight,
       RVARFMT=6.1,    /* make wide enough for total */
       RVARLABL=Weight in Pounds,
       NN=10, MINRVAR=.,
       TTLTEXT=Students By Weight)
run;
```

Output 3B:

```
Ranked List of Students By Weight

This list accounts for 21.6% of the total Weight in Pounds

        Only values not less than 125.0 are listed

                                Weight
                                  in
                Rank    Student   Pounds

                  1     Philip    150.0
                  2     Ronald    133.0
                  3     Robert    128.0
                                 ======
                                  411.0
```

Program 3B:

```
%TOPNN(DATA=sashelp.class,
       CLASSVAR=Name, CVARLABL=Student,
       RANKVAR=Weight,
       RVARFMT=6.1,    /* make wide enough for total */
       RVARLABL=Weight in Pounds,
       NN=10, MINRVAR=125,
       TTLTEXT=Students By Weight)
run;
```

Macro Used to Produce Outputs 3A & 3B:

```
%MACRO TOPNN
(DATA=,
 CLASSVAR=,
 CVARLABL=,
 RANKVAR=,
 RVARFMT=,
 RVARLABL=,
 NN=,
 MINRVAR=.,
 TTLTEXT=);

DATA FORTOPNN;
SET &DATA;
%GLOBAL BELOWMIN;
IF _N_ EQ 1 THEN CALL SYMPUT('BELOWMIN','N');
IF &MINRVAR NE . THEN DO;
  IF &RANKVAR GE &MINRVAR THEN RETURN;
  ELSE DO;
    CALL SYMPUT('BELOWMIN','Y');
    DELETE;
  END;
END;
KEEP &CLASSVAR &RANKVAR;
RUN;

PROC SORT OUT=FORTOPNN;
BY DESCENDING &RANKVAR;
RUN;

DATA TOREPORT;
SET FORTOPNN;
IF _N_ LT &NN + 1;
RANK = _N_;
RUN;

PROC MEANS DATA=&DATA NOPRINT SUM N;
VAR &RANKVAR;
OUTPUT OUT=ALL SUM=SUMTOT N=NTOT;
RUN;

PROC MEANS DATA=TOREPORT NOPRINT SUM N;
VAR &RANKVAR;
OUTPUT OUT=TOPNN SUM=SUMTOP N=NTOP;
RUN;
```

```
DATA _NULL_;
MERGE ALL TOPNN;
FORMAT PCTTOT 5.1;
PCTTOT = ROUND((100 * (SUMTOP / SUMTOT)),.1);
%GLOBAL RANKLEN;
CALL SYMPUT('RANKLEN',LENGTH(LEFT(&NN)));
%GLOBAL MIN;
IF &MINRVAR NE . THEN CALL
SYMPUT('MIN',TRIM(LEFT(PUT(&MINRVAR,&RVARFMT))));
%GLOBAL PCTTOT;
CALL SYMPUT('PCTTOT',TRIM(LEFT(PCTTOT)));
%GLOBAL HTTLMIN;
FORMAT TTLTOPNN $14.;
IF NTOP LT NTOT AND NTOP EQ &NN THEN DO;
    TTLTOPNN = "Top &NN";
    CALL SYMPUT('HTTLMIN','0');
  END;
ELSE DO;
  TTLTOPNN = 'Ranked List of';
  IF &MINRVAR EQ . OR "&BELOWMIN" EQ 'N'
    THEN CALL SYMPUT('HTTLMIN','0');
  ELSE   CALL SYMPUT('HTTLMIN','1');
END;
%GLOBAL TTLTOPNN;
CALL SYMPUT('TTLTOPNN',TRIM(TTLTOPNN));
RUN;

OPTIONS NODATE NONUMBER;
PROC PRINT DATA=TOREPORT NOOBS U LABEL SPLIT='*';
FORMAT RANK     &RANKLEN..;
FORMAT &RANKVAR &RVARFMT;
LABEL RANK      = 'Rank'
      &CLASSVAR = "&CVARLABL"
      &RANKVAR  = "&RVARLABL";
VAR RANK &CLASSVAR &RANKVAR;
SUM &RANKVAR;
TITLE1 "&TTLTOPNN &TTLTEXT";
TITLE3 "This list accounts for &PCTTOT% of the total &RVARLABL";
%IF &HTTLMIN EQ 1 %THEN %DO;
TITLE5 "Only values not less than &MIN are listed";
%END;

%MEND TOPNN;
```

Example 4. Produce an Exception Report, but be able to automatically handle the situation of No Exceptions. (A valuable extra is to let the program send email. One option is to email both good and bad news; another option is to email only bad news. Another choice is whether to attach the exception report to the email, or to web publish it and just email a hyperlink. Neither of these enhancements is covered here, but they were implemented in Reference 5 for an Enterprise Performance Reporting system.)

Output 4A.

```
 Tall Students in SASHELP.CLASS
Exception Report for Height GT 65

    Obs    Name       Height

     1     Barbara     65.3
     2     Mary        66.5
     3     Alfred      69.0
     4     Philip      72.0
     5     Ronald      67.0
     6     William     66.5
```

Program 4A.

```
%ReportOneTypeOfExceptions
      (data=sashelp.class,
       var=Height,
       comparison=GT,
       standard=65,
       ExceptionIDvar=Name,
       title=Tall Students in SASHELP.CLASS)
run;
```

Output 4B.

```
 Heavy Students in SASHELP.CLASS
Exception Report for Weight GT 150

     No Exceptions To Report
```

Program 4B.

```
%ReportOneTypeOfExceptions
      (data=sashelp.class,
       var=Weight,
       comparison=GT,
       standard=150,
       ExceptionIDvar=Name,
       title=Heavy Students in SASHELP.CLASS)
run;
```

Macro Used to Produce Outputs 4A & 4B:

```
%macro ReportOneTypeOfExceptions
      (data=,
       var=,
       comparison=,
       standard=,
       ExceptionIDvar=,
       title=);

%let ExceptionsFound = N;

data
  Exceptions  (keep=&ExceptionIDvar &var)
  NoExceptions(keep=Message);
retain Message 'No Exceptions To Report';
set &data;
if _N_ EQ 1 then output NoExceptions;
if &var &comparison &standard;
call symput('ExceptionsFound','Y');
output Exceptions;
run;

title1 ' '; /* spacer: this could be eliminated */
title2 "&title";
title3 "Exception Report for &var &comparison &standard";

%if &ExceptionsFound EQ Y
%then %do;

proc print data=Exceptions;
var &ExceptionIDvar &var;

%end;

%else %do;

proc print data=NoExceptions noobs label;
label Message='00'X; /* Message needs no report column label */
var Message;

%end;

run;

%mend  ReportOneTypeOfExceptions;
```

Example 5. Produce an Exception Report that covers multiple types of exceptions. As types of exceptions are added to and removed from the report, and/or as the standard for any particular exception is changed over time, no program change must be required. Create the data analyzer / report writer once, and never change it. First, let us look at the output and the coding, and at part of the SAS log from the execution of Program 5A. Then, we will look at the control table.

(This example is really derivative of part of a Software-Intelligent application for Enterprise Performance Reporting prototype in Reference 5. That paper was an update, to the era of web and email, based on my first SI application development in 1987, when I built a performance, capacity, and usage reporting system for IT computer and communications resources.)

Output 5A:

Exceptions in SASHELP.CLASS using Criteria Set 1

Obs	Student	Exception	Variable	Value	Comparison	Standard
1	Barbara	Tall	Height	65.3	GT	65.0
2	Mary	Tall	Height	66.5	GT	65.0
3	Alfred	Tall	Height	69.0	GT	65.0
4	Philip	Tall	Height	72.0	GT	65.0
5	Philip	Heavy	Weight	150.0	GT	135.0
6	Ronald	Tall	Height	67.0	GT	65.0
7	William	Tall	Height	66.5	GT	65.0

Program 5A.

```
%ReportMultipleTypesOfExceptions
(data=sashelp.class,
 criteria=Define.ExceptionCriteria1,
 ExceptionIDvar=Name,
 ExceptionIDvarLabel=Student,
 title=Exceptions in SASHELP.CLASS using Criteria Set 1)
run;
```

Output 5B:

Exceptions in SASHELP.CLASS using Criteria Set 2

No Exceptions Found

Program 5B.

```
%ReportMultipleTypesOfExceptions
(data=sashelp.class,
 criteria=Define.ExceptionCriteria2,
 ExceptionIDvar=Name,
 ExceptionIDvarLabel=Student,
 title=Exceptions in SASHELP.CLASS using Criteria Set 2)
run;
```

Macro Used to Produce Outputs 5A & 5B:

```
%macro ReportMultipleTypesOfExceptions
      (data=,
       criteria=,
       title=,
       ExceptionIDvar=,
       ExceptionIDvarLabel=);

data _null_;
set &criteria end=LastOne;
call symput('ExcDesc'||trim(left(_N_)),trim(left(ExceptionDesc   )));
call symput('ExcVar' ||trim(left(_N_)),trim(left(ExceptionVar    )));
call symput('ExcComp'||trim(left(_N_)),trim(left(ExceptionCompare)));
call symput('ExcStd' ||trim(left(_N_)),trim(left(ExceptionStd    )));
call symput('ExcFmt' ||trim(left(_N_)),trim(left(ExceptionFormat )));
if LastOne;
call symput('NumberOfCriteria',_N_);
run;

data ExceptionsFound;
keep &ExceptionIDvar ExcDesc ExcVar ExcValue ExcComp ExcStd;
label  &ExceptionIDvar = "&ExceptionIDvarLabel";
label  ExcDesc  = 'Exception';
label  ExcVar   = 'Variable';
label  ExcValue = 'Value';
label  ExcComp  = 'Comparison';
label  ExcStd   = 'Standard';
length ExcDesc  $ 50;
length ExcVar   $ 50;
length ExcValue $ 50;
length ExcComp  $  3;
length ExcStd   $ 50;
set &data;

%do i = 1 %to &NumberOfCriteria;
if &&ExcVar&i &&ExcComp&i &&ExcStd&I then do;
  ExcDesc  = "&&ExcDesc&i";
  ExcVar   = "&&ExcVar&i";
  ExcValue = put(&&ExcVar&i,&&ExcFmt&i);
  ExcComp  = "&&ExcComp&i";
  ExcStd   = put(&&ExcStd&i,&&ExcFmt&i);
  output;
end;
%end;

run;
```

```
data _null_;

if NumberOfExceptions NE 0
then do;

  call symput('ExceptionsFound','Y');
  stop;

end;

else do;

  call symput('ExceptionsFound','N');
  Message = 'No Exceptions Found';
  output;

end;

set ExceptionsFound nobs=NumberOfExceptions;
run;

title1 ' ';
title2 "&title";

%if &ExceptionsFound EQ Y
%then %do;

proc print data=ExceptionsFound label;

%end;

%else %do;

proc print data=NoExceptions noobs label;
label Message='00'X;
var Message;

%end;

run;

%mend  ReportMultipleTypesOfExceptions;
```

SAS Log Excerpt to Show Run-Time Generated Code for ExceptionsFound DATA Step for Program 5A, where criteria=Define.ExceptionCriteria1:

```
data ExceptionsFound;

keep Name ExcDesc ExcVar ExcValue ExcComp ExcStd;

label Name = "Student";
label ExcDesc = 'Exception';
label ExcVar = 'Variable';
label ExcValue = 'Value';
label ExcComp = 'Comparison';
label ExcStd = 'Standard';

length ExcDesc $ 50;
length ExcVar $ 50;
length ExcValue $ 50;
length ExcComp $ 3;
length ExcStd $ 50;

set sashelp.class;

if Height GT 65 then do;
  ExcDesc = "Tall";
  ExcVar = "Height";
  ExcValue = put(Height,4.1);
  ExcComp = "GT";
  ExcStd = put(65,4.1);
  output;
end;

if Weight GT 135 then do;
  ExcDesc = "Heavy";
  ExcVar = "Weight";
  ExcValue = put(Weight,5.1);
  ExcComp = "GT";
  ExcStd = put(135,5.1);
  output;
end;

run;

 /* The blank lines and indents were added by the author after
pasting this code in from the SAS log. Macro generated code is
displayed in the SAS log only if using OPTIONS MPRINT. The macro name
prefix that appears at the left end of each code line in the log was
erased by the author after pasting this code in from the log. */
```

Control Table Used by Program 5A. This DATA Step is for illustration only. You could support the control table with **View/Edit Table** in a SAS Display Manager session, or could support the control data as a .txt file or .csv file, either of which can be read with a SAS program. Instead of this DATA Step being used to create the Exception Definitions as a SAS data set, it could be replaced by one that reads such an external .txt or .csv file to create a SAS data set for input to the macro. It should not require a SAS programmer to maintain the control table. That is an administrative task.

```
libname Define 'C:\ExceptionDefinitions';

data Define.ExceptionCriteria1;

 /* five variables for each Exception Criterion */

label  ExceptionDesc    = 'Exception Description';
label  ExceptionVar     = 'Exception Variable';
label  ExceptionCompare = 'Exception Comparison';
label  ExceptionStd     = 'Standard Value';
label  ExceptionFormat  = 'Format for Data Values';

length ExceptionDesc      $ 50;
length ExceptionVar       $ 50;
length ExceptionCompare   $  3;
length ExceptionStd          8;
length ExceptionFormat    $ 40;

 /* Define each Exception Criterion */

ExceptionDesc    = 'Tall';
ExceptionVar     = 'Height';
ExceptionCompare = 'GT';
ExceptionStd     = 65.0;
ExceptionFormat  = '4.1';
output;

ExceptionDesc    = 'Heavy';
ExceptionVar     = 'Weight';
ExceptionCompare = 'GT';
ExceptionStd     = 135.0;
ExceptionFormat  = '5.1';
output;

       /* Repeat assignments and output statement
          for each Exception Variable.
          Add, change, delete at any time. */
run;
       /* To build Define.ExceptionCriteria2, in code above
          simply replace 65 with 95 and 135 with 155. */
```